5177816

Performance Working Group
Grid Working Document: GWD-Perf-8-2
Category: Informational
Obsoletes: GWD-Perf-1-1

Warren Smith
NASA Ames Research Center
Dan Gunter
Lawrence Berkeley National Laboratory
Darcy Quesnel
Argonne National Laboratory
June 6 2001

A Simple XML Producer-Consumer Protocol

1. Introduction

There are many different projects from government, academia, and industry that provide services for delivering events in distributed environments. The problem with these event services is that they are not general enough to support all uses and they speak different protocols so that they cannot interoperate. We require such interoperability when we, for example, wish to analyze the performance of an application in a distributed environment. Such an analysis might require performance information from the application, computer systems, networks, and scientific instruments. In this work we propose and evaluate a standard XML-based protocol for the transmission of events in distributed systems.

One recent trend in government and academic research is the development and deployment of computational grids [14]. Computational grids are large-scale distributed systems that typically consist of high-performance compute, storage, and networking resources. Examples of such computational grids are the DOE Science Grid [3], the NASA Information Power Grid (IPG) [8, 18], and the NSF Partnerships for Advanced Computing Infrastructure (PACIs) [9, 10]. The major effort to deploy these grids is in the area of developing the software services to allow users to execute applications on these large and diverse sets of resources. These services include security, execution of remote applications, managing remote data, access to information about resources and services, and so on. There are several toolkits for providing these services such as Globus [4, 13], Legion [7, 15], and Condor [1, 19].

As part of these efforts to develop computational grids, the Global Grid Forum [5] is working to standardize the protocols and APIs used by various grid services. This standardization will allow interoperability between the client and server software of the toolkits that are providing the grid services. The goal of the Performance Working Group [6] of the Grid Forum is to standardize protocols and representations related to the storage and distribution of performance data. These standard protocols and representations must support tasks such as profiling parallel applications, monitoring the status of computers and networks, and monitoring the performance of services provided by a computational grid.

This paper describes a proposed protocol and data representation for the exchange of events in a distributed system. The protocol exchanges messages formatted in XML and it can be layered atop any low-level communication protocol such as TCP or UDP. Further, we describe Java and C++ implementations of this protocol and discuss their performance.

The next section will provide some further background information. Section 3 describes the main communication patterns of our protocol. Section 4 describes how we represent events and related information using XML. Section 5 describes our protocol and Section 6 discusses the performance of two implementations of the protocol. Finally, an appendix provides the XML Schema definition of our protocol and event information.

2. Background

The Grid Forum Performance Working Group has defined the basic architecture shown in Figure 1. This architecture consists of three components: a producer, a consumer, and a directory service. A *producer* is something that is producing performance data, each unit of which is called an *event*. This producer can be an application profiler, a host monitor, or anything else. A consumer is something that consumes or receives events. A *consumer* might be a tool to calculate how much time is spent in each function of an application or a graphical interface showing the status of a set of hosts. A *directory service* is a type of database that is used to store and retrieve information about the producers and consumers, which is accessed using a protocol such as the Lightweight Directory Access Protocol (LDAP) [17]. A producer may advertise a host monitor in the directory service so that a consumer can search the directory service and find the monitor for a certain host. The consumer can than contact that producer in order to receive events about that host.

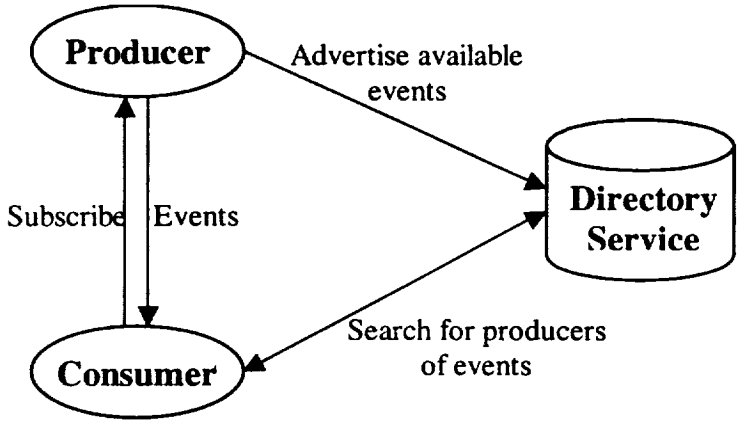

Figure 1. Grid Monitoring Architecture.

The Grid Forum Performance Working Group is defining the protocols and data representations required by this architecture. This includes:

- A definition of the structure and organization of the data in the directory service,
- A definition of events and information related to events, and
- The protocol for communicating between producers and consumers of events.

In this paper we describe a proposed producer-consumer communication protocol and the event information that is required by this protocol. Our protocol consists of a XML encoding of messages and the state machines that describe when these messages are sent. We do not specify the transport protocol on top of which our protocol will be layered. The transport protocol could be UDP, TCP, HTTP, SSL, or any number of other protocols. We choose to use XML to represent our data for several reasons. First, XML provides a textual representation of data that is readable and therefore easier to debug. We could have selected a binary representation of our data for improved performance, but a textual approach seems more appropriate at our current experimental stage. Second, XML is self-describing and hierarchical, which makes it easy to represent structured event data. Third, XML was selected instead of any other textual representation because of the large and growing number of XML tools available and the growing number of people familiar with XML.

Another approach we could have taken was to use SOAP [12] or XML-RPC [11] and thus avoid explicit representation of the XML for each message. While this approach is a valid one, it has several drawbacks. First, neither SOAP nor XML-RPC has low-level transport bindings: SOAP has HTTP and SNMP bindings, and XML-RPC has only an HTTP binding. This introduces an additional layer of inefficiency and, more importantly, makes it difficult to return a stream of information to a consumer through a firewall. At some sites, inability to operate through a firewall could in and of itself make the protocol useless. However, because of the convenience of these higher-level tools, particularly for relatively performance-insensitive "control" messages, we are considering a possible convergence in future versions of the protocol that solves this problem.

Another approach would have been to use CORBA [20], for instance the CORBA Event Service. This approach, while also valid, would impose a significant administrative and development overhead if the target community did not already use CORBA, as is the case with the academic and scientific communities. SNMP [21] was also considered, but was not used due to its inability to handle streaming data efficiently.

3. Interactions

There are three major classes of producer-consumer interactions that our protocol must support. As interactions with the directory service are outside the scope of this document, it is assumed that in each interaction, producers and consumers are able to locate each other and determine which events the other can produce or consume.

Figure 1 shows our first interaction. In the figure, a consumer subscribes to specific events from the producer. Then the producer sends the events the consumer subscribed for to the consumer. These events are sent out over a period of time until the producer or consumer (the consumer, in the figure) ends the subscription. We call this interaction a *consumer-initiated subscription* or simply consumer subscribe.

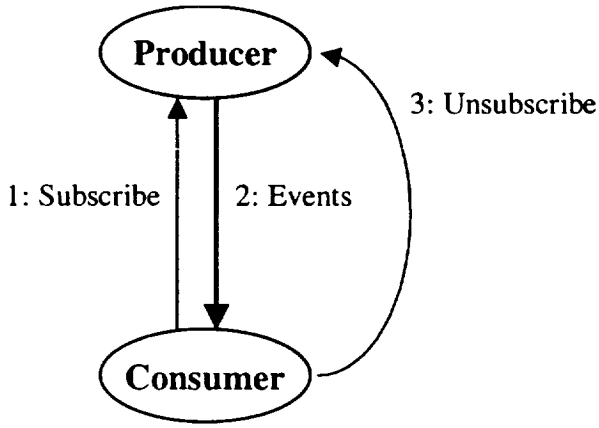

Figure 1. Consumer-initiated subscription.

The second type of interaction is the *producer-initiated subscription*, or simply producer subscribe. This interaction is shown in Figure . First, the producer contacts a consumer to request a subscription. Then events are sent from the producer to the consumer until the subscription is

terminated. This type of interaction is useful, for example, when a producer sends events to an archive. In this case, the archive is the consumer.

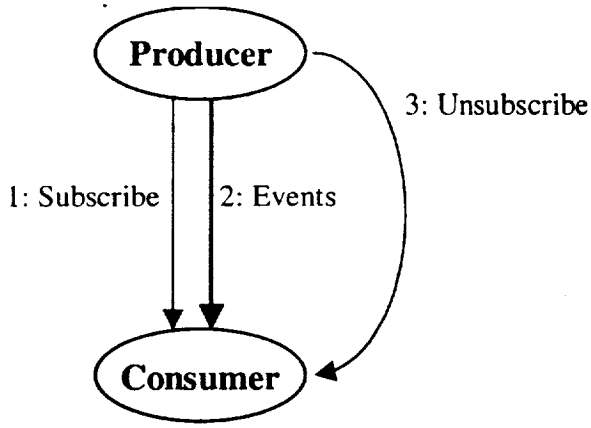

Figure 2. Producer-initiated subscription.

The third type of interaction is a simple *request/reply* that is shown in Figure 3. The figure shows the consumer requesting information from a consumer, but a producer can also make a request of a consumer. Our two previous interactions include request/reply interactions but our protocol includes two instances of this interaction that stand on their own. First, there is a *query* interaction. In this interaction the consumer queries a producer for a single event and the producer replies with the event. Second, there is an *event names* interaction where a consumer requests a list of the events available from a producer and the producer replies with the list.

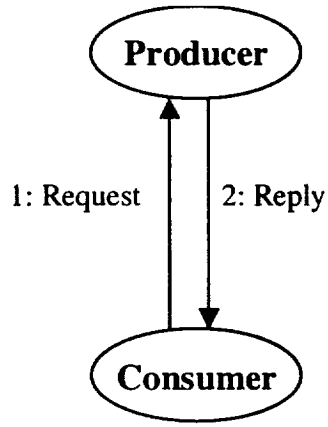

Figure 3. Request/reply interaction

4. Events and Event Parameters

Before we describe our protocol, we first describe how we use XML to represent events and event parameters. In this section and the following sections we provide example XML representations of the data we are representing. Appendix A provides the XML Schema for our data.

As mentioned before, events are the basic unit of information in our architecture. An event is a set of <name, value> pairs where the values are typed and there is always a pair that contains the time the event was generated. We represent this time using a time stamp that is a string formatted according to the proposed Grid Forum standard format [16] This format is an extension of the ISO 8601 time format [2]. Each element also has two optional attributes: units and accuracy. The units attribute indicates the units associated with the element's value (e.g.: 'degrees', or 'bytes') and the accuracy attribute indicates what range of likely "real" values are represented by the element's value (e.g. '+/-5.0').

Associated with each event is a set of parameters that describe the information that can be passed to a producer of events as part of a subscription or query. The event parameters consist of a set of <name, value> pairs. Each element can have a units attribute associated with it. Examples of event parameters are shown in Section 4.1 and Section 0.

4.1. CPU Load Event

The CPU load event is a simple event for containing the load information returned by the Unix uptime command. We therefore use the event type "UptimeCPULoad" for this event to differentiate it from other means of measuring CPU load. This event must contain the following elements:

- TimeStamp. The time at which the CPU load event was generated.
- Load1. The 1 minute CPU load reported by uptime.
- Load5. The 5 minute CPU load reported by uptime.
- Load15. The 15 minute CPU load reported by uptime.
- HostName. The name of the host the load measurement is made on.

Here is an example of such an event in our XML encoding:

```
<UptimeCPULoad xmlns="http://www.gridforum.org/Performance/Events">
    <Load1>1.5</Load1>
    <Load5>1.6</Load5>
    <Load15>1.3</Load15>
    <HostName>foo.nas.nasa.gov</HostName>
    <TimeStamp>2000-11-09T21:51:45Z</TimeStamp>
</UptimeCPULoad>
```

Note that we define a namespace for all events defined by the Grid Forum Performance Working Group. When asking for a CPU load event, the following input parameters can be specified:

- Period. The number of seconds between each uptime event generation. This parameter is only used when a subscription is performed. If this parameter is specified for a query, it is ignored.

An example of the parameters that can be specified for this event is:

```
<UptimeCPULoad
xmlns="http://www.gridforum.org/Performance/EventParameters">
    <Period units="min">600</Period>
</UptimeCPULoad>
```

Again note that we have defined a namespace for the parameters that can be specified when requesting events defined by the Grid Forum Performance Working Group.

4.2. Round Trip Time Event

The second event we define here is a network latency event with data produced by the Unix ping command. We simply call this event "Ping". The event must contain the following elements:

- TimeStamp. The time at which the ping was performed.
- SourceHostName. The host name or IP address of the host that is performing the ping command.
- TargetHostName. The host name or IP address of the host that the source host is pinging. Note that this name or IP address can indicate 1 of several network interfaces on the target host.
- RoundTripTime. The round-trip time reported by the ping command. The default units for this value are milliseconds.

Here is an example of such an event in our XML encoding:

```
<Ping xmlns="http://www.gridforum.org/Performance/Events">
    <SourceHostName>foo.nas.nasa.gov</SourceHostName>
    <TargetHostName>bar.lbl.gov</TargetHostName>
    <RoundTripTime>7</RoundTripTime>
    <TimeStamp>2000-11-09T21:53:45Z</TimeStamp>
</Ping>
```

When asking for a ping event, the following input parameters can be specified:

- Period. The number of seconds between each uptime measurement and event generation. This parameter is only used when a subscription is performed. If this parameter is specified for a query, it is ignored.
- TargetHostName. The name or IP address of the host that will be pinged. This parameter is required.

An example of the parameters that can be specified when asking for a ping event are:

```
<Ping xmlns="http://www.gridforum.org/Performance/EventParameters">
    <Period>600</Period>
    <TargetHostName>bar.lbl.gov</TargetHostName>
</Ping>
```

5. Protocol

This section describes the XML protocol we use for communication between producers and consumers. A formal definition of this protocol is provided in XML Schema in Appendix A. Our protocol supports all of the interactions described in Section 3. We begin by discussing some general formatting issues.

5.1. General Message Format

In general, each message consists of:

1. The number of bytes in the message. For our TCP binding, this is a 32-bit integer in network byte order.
2. The XML tags that indicate the message type.

3. Request messages always have a requester-unique request ID chosen by the requestor. This request ID is an attribute of the message tag
4. Reply messages always have a request ID, which matches the request ID of the request that is being replied to.
5. Reply messages always have a return code and may have a detailed return message. The Return element indicates if an operation was successful (Success) or a failure (Failure). These return codes will most likely be expanded later to contain more detailed error codes. The ReturnDetail element contains a text message that contains detailed user-readable information about the status of a request.
6. The message-specific data inside the XML tags that identify the message.

We define three XML name spaces for use in our protocol. The name space *http://www.gridforum.org/Performance/Events* contains the events defined by the Grid Forum Performance Working Group, the name space *http://www.gridforum.org/Performance/EventParameters* contains the parameters defined by the working group that can be specified when asking for an event or events, and the name space *http://www.gridforum.org/Performance/Protocol* contains the elements which make up the messages of our protocol. Further, we allow any group to define events and event parameters in their own name spaces for use with our protocol.

5.2. Consumer-Initiated Subscription

When a consumer wants to receive a stream of events from a producer, it subscribes to the producer for the events. After a subscription successfully takes place, events are sent from the producer to the consumer until either the consumer or producer unsubscribes. There are five messages in this process and these messages and the associated state machine are shown in Figure 3. If there is no label for a state transition, we assume that the transition always occurs if no errors have occurred during the actions taken in a state. To make our state machines easier to read, we do not show a failure state that is entered if any operation fails.

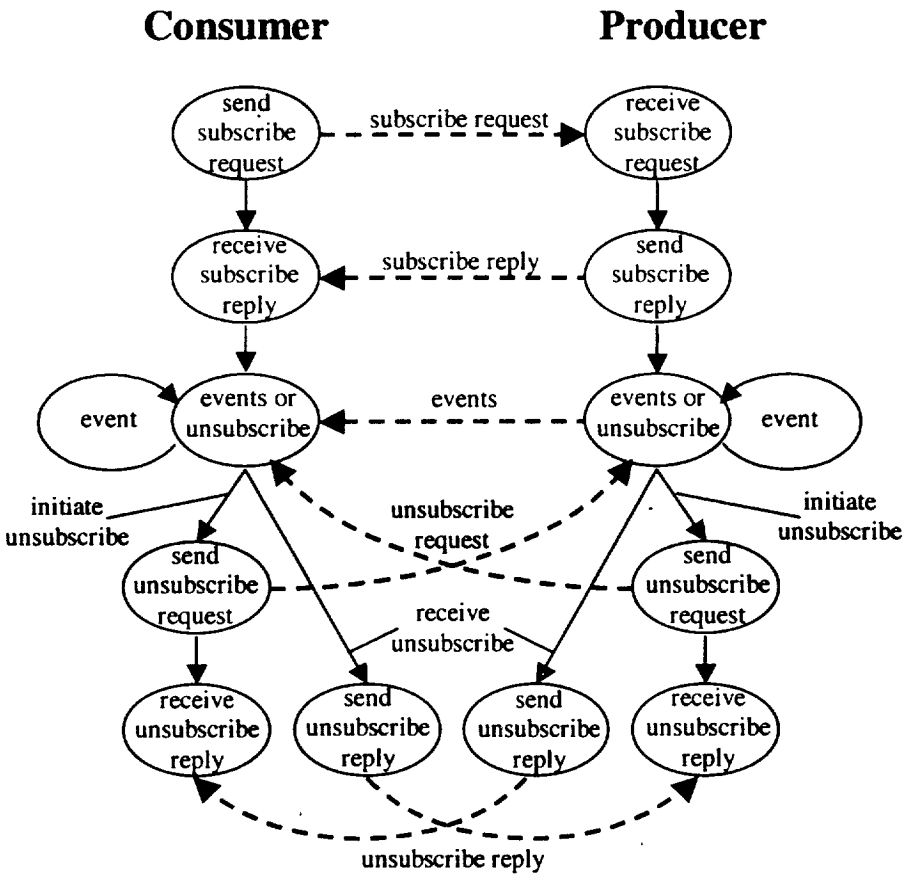

Figure 3. The state machine and messages for a consumer-initiated subscription.

5.2.1. Subscribe Request

The subscribe request message consists of:

- A consumer-unique request ID that the reply to this message will refer to (required).
- A consumer-unique subscription ID that the consumer will use to identify the subscription (required).
- Event parameters element (required).
- Any input parameters needed to generate events (optional).

Here are two examples of subscribe request messages:

```
<SubscribeRequest xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="1">
  <SubscriptionID>12</SubscriptionID>
    <UptimeCPULoad
xmlns="http://www.gridforum.org/Performance/EventParameters">
        <Period>600</Period>
    </UptimeCPULoad>
</SubscribeRequest>
```

```
<SubscribeRequest xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="2">
    <SubscriptionID>13</SubscriptionID>
    <Ping xmlns="http://www.gridforum.org/Performance/EventParameters">
        <Period>300</Period>
        <TargetHostName>bar.lbl.gov</TargetHostName>
    </Ping>
</SubscribeRequest>
```

In the future, we will add an optional event filter to subscription request messages. The filter will select events from the entire set of events generated by the producer for the subscription. For example, a filter may indicate that only CPU load events with a 1-minute and 5-minute load average greater than or equal to 5.0 should be sent. There are many possibilities for the filter language, but none is clearly superior. One standard filter that is in widespread use is the LDAP filter language [17]. This language provides a logical expression of what in LDAP are called "attribute" values, but what we would call element values. The syntax is a prefix notation using only a few operators (and, or, not, greater than or equal to, less than or equal to, equals, approximately equal) and a wildcard character for partial string matching. A filter string for an UptimeCPULoadEvent that would implement the CPU load criteria mentioned above would be: `(|(Load1 >= 5)(Load5 >= 5))`.

5.2.2. Subscribe Reply

The subscribe reply message consists of:

- The requestID (required) of the request that this message is in reply to.
- Return (required). Success means the request the reply is for was successfully completed, Failure means the request failed. Other return codes to represent more detailed failures will most likely be added in the future.
- ReturnDetail (optional). Text giving further information about the successful or unsuccessful subscribe.
- An optional producer-unique SubscriptionID that identifies the subscription that was successfully made by the consumer (if one was). The subscription ID should be present if the subscription was successful and should not be present if the subscription was not successful. The SubscriptionID can be used to unsubscribe later.

Two examples of subscribe reply messages are:

```
<SubscribeReply xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="1">
    <Return>Failure</Return>
    <ReturnDetail>The period specified is too small.
    </ReturnDetail>
</SubscribeReply>

<SubscribeReply xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="2">
    <Return>Success</Return>
    <SubscriptionID>15</SubscriptionID>
</SubscribeReply>
```

5.2.3. Unsubscribe Request

Unsubscribe requests can originate at either the producer or consumer. In either case, the message has the same format. The unsubscribe request message consists of:

- The requestID (required) of this request so that the reply to this request can be identified.
- The SubscriptionID (required) generated by the message target (i.e. producer if the originator is the consumer, consumer if the originator is the producer) that identifies the subscription that is being terminated.

An example of an unsubscribe request message is:

```
<UnsubscribeRequest
xmlns="http://www.gridforum.org/Performance/Protocol" requestID="9">
    <SubscriptionID>1234</SubscriptionID>
</UnsubscribeRequest>
```

5.2.4. Unsubscribe Reply

The unsubscribe reply message consists of:

- The requestID (required) of the request that this message is in reply to.
- Return (required). Success means the request the reply is for was successfully completed, Failure means the request failed. Other non-zero return codes to represent more detailed failures will most likely be added in the future.
- ReturnDetail (optional). This elements provides text giving further information about the successful or unsuccessful unsubscribe.

Examples of unsubscribe reply message are:

```
<UnsubscribeReply xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="9">
    <Return>Success</Return>
</UnsubscribeReply>

<UnsubscribeReply xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="9">
    <Return>Failure</Return>
    <ReturnDetail>Unknown subscription ID.</ReturnDetail>
</UnsubscribeReply>
```

5.2.5. Event

An event message is sent from the producer to the consumer after a subscription is initiateds. An event message consists of:

- The subscription ID (required) that was generated by the consumer which identifies which subscription the event belongs to.
- The event (optional) in the format described in Section 4. The event should be present if an error is not reported.
- Error (optional), indicating that an error occurred while generating the event.
- ErrorDetail (optional) providing further information about an error that occurred while generating an event. This element should only occur in conjunction with the Error element.

Example event messages are shown below. Note that the event elements are in the gfperf-event namespace.

```
<Event xmlns="http://www.gridforum.org/Performance/Protocol"
subscriptionID="1234">
    <UptimeCPULoad xmlns="http://www.gridforum.org/Performance/Events">
```

```
        <Load1>1.5</Load1>
        <Load5>1.6</Load5>
        <Load15>1.3</Load15>
        <TimeStamp>2000-11-09T21:51:45Z</TimeStamp>
    </UptimeCPULoad>
</Event>

<Event xmlns="http://www.gridforum.org/Performance/Protocol"
subscriptionID="1234">
    <UptimeCPULoad xmlns="http://www.gridforum.org/Performance/Events">
        <Error>Authorization</Error>
        <ErrorDetail>You are no longer authorized to receive this
                    information.</ErrorDetail>
    </UptimeCPULoad>
</Event>


<Event xmlns="http://www.gridforum.org/Performance/Protocol"
subscriptionID="1235">
    <Ping xmlns="http://www.gridforum.org/Performance/Events">
        <SourceHostName>foo.nas.nasa.gov</SourceHostName>
        <TargetHostName>bar.lbl.gov</TargetHostName>
        <RoundTripTime>7</RoundTripTime>
        <TimeStamp>2000-11-09T21:53:45Z</TimeStamp>
    </Ping>
</Event>
```

5.3. Producer-Initiated Subscription

There are cases where a producer of events may want to initiate a subscription. A common case is when a producer wants to archive the events it is generating. The state machine and messages for this type of interaction are shown in Figure 4. The request and reply messages used during a producer-initiated subscription are identical to those used for a consumer-initiated subscription. The only difference is that the producer is requesting the subscription instead of the consumer.

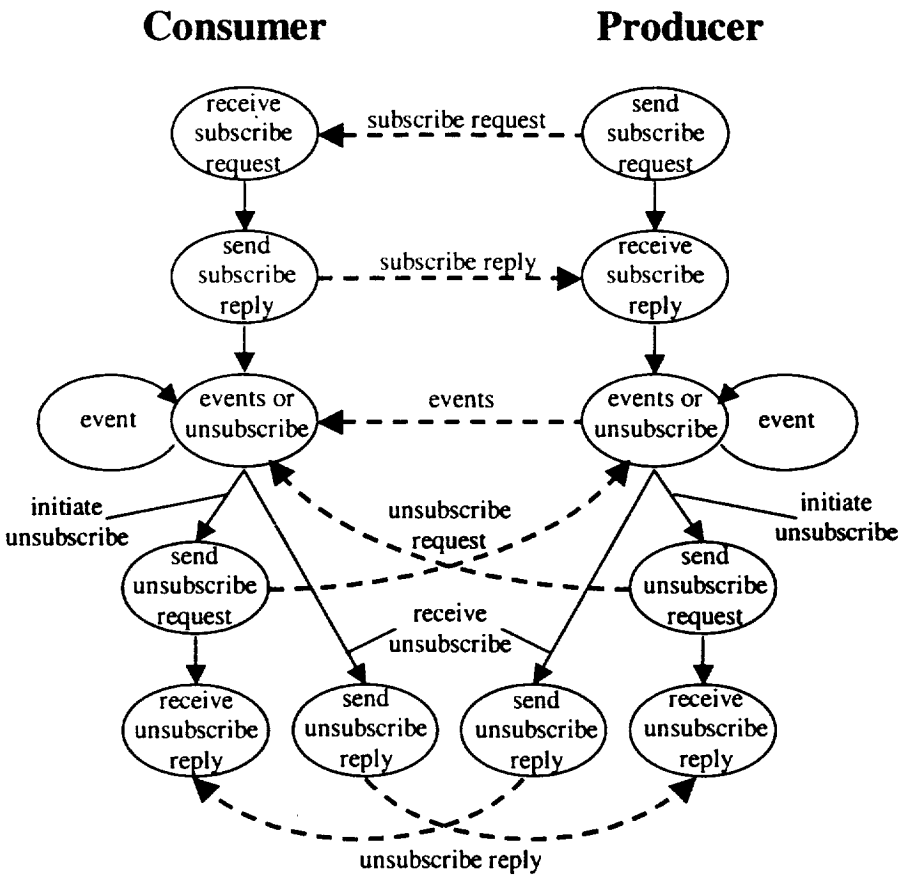

Figure 4. The state machine and messages for a producer-initiated subscription.

5.4. Querying for an Event

Often a consumer will want just 1 event from a producer. Instead of having a consumer subscribe, receive 1 event, and then unsubscribe, we allow a consumer to query a producer for an event. The state machine and messages for this example of the request/reply interaction are shown in Figure 5. A query consists of a query request message that a consumer sends to the producer and a query reply message that the producer sends to the consumer in response to the query request message. The query reply includes the event that was requested.

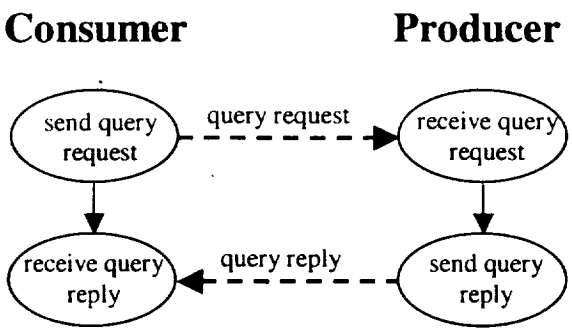

Figure 5. The state machine and messages for a query.

5.4.1. Query Request

The query request message is very similar to the consumer subscribe request message and consists of:

- A request ID (required).
- Event parameters element (required).
- Any input parameters needed to generate events (optional).

Here are two examples of QueryRequest messages:

```
<QueryRequest xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="15">
    <UptimeCPULoad
xmlns="http://www.gridforum.org/Performance/EventParameters"/>
</QueryRequest>

<QueryRequest xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="20">
    <Ping xmlns="http://www.gridforum.org/Performance/EventParameters">
        <TargetHostName>bar.lbl.gov</TargetHostName>
    </Ping>
</QueryRequest>
```

5.4.2. Query Reply

The query reply messages are similar to the event messages and consist of:

- A request ID (required).
- Return (required).
- ReturnDetail (optional).
- The event data in the format described in Section 4.

Example query reply messages are:

```
<QueryReply xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="15">
    <Return>Success</Return>
    <UptimeCPULoadEvent
xmlns="http://www.gridforum.org/Performance/Events">
        <Load1>1.5</Load1>
        <Load5>1.6</Load5>
```

```
        <Load15>1.3</Load15>
        <TimeStamp>2000-11-09T21:51:45Z</TimeStamp>
    </UptimeCPULoadEvent>
</QueryReply>


<QueryReply xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="20">
    <Return>Success</Return>
    <PingEvent xmlns="http://www.gridforum.org/Performance/Events">
        <SourceHostName>foo.nas.nasa.gov</SourceHostName>
        <TargetHostName>bar.lbl.gov</gfperf-event:TargetHostName>
        <gfperf-event:RoundTripTime>7</RoundTripTime>
        <TimeStamp>2000-11-09T21:53:45Z </TimeStamp>
    </PingEvent>
</QueryReply>
```

5.5. Requesting Available Events

Even though our architecture in Figure 1 shows a directory service that will be used to contain information on the events that are available from a producer it is also convenient to be able to obtain this information from producers directly. This message sequence, shown in Figure 6, is an example of a request/reply interaction.

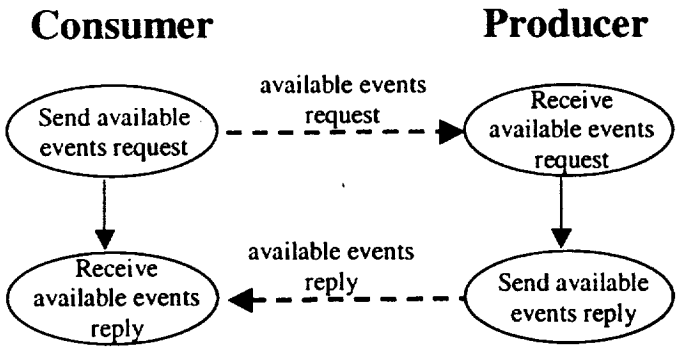

Figure 6. The state machine and messages for requesting available events

5.5.1. Event Names Request

The available events request message is very simple and only contains a request ID. Here is an example EventNamesRequestMessage:

```
<EventNamesRequest
xmlns="http://www.gridforum.org/Performance/Protocol" requestID="15"/>
```

5.5.2. Event Names Reply

The event names reply messages consist of:

- A request ID (required).
- Return (required).
- ReturnDetail (optional).

- One or more Event elements the value of which is the name of the event. This element also has two attributes:
  - The eventns attribute specifies the name space of the event.
  - The paramns attribute specifies the name space of the parameters to the event.

An example event names reply messages is shown below.

```
<EventNamesReply xmlns="http://www.gridforum.org/Performance/Protocol"
requestID="15">
    <Return>Success</Return>
    <Event eventns="http://www.gridforum.org/Performance/Events"
        paramns="http://www.gridforum.org/Performance/EventParameters">
        UptimeCPULoad
    </Event>
</EventNamesReply>
```

6. Performance

In this section we present performance results for two independent implementations of our protocol. One implementation uses Java and the Xerces XML parser. The other implementation uses C++ and the expat XML parser. We examined the performance of these implementations using a 933 MHz Pentium III system running RedHat Linux 7.1 with JDK 1.3. We found that the C++ implementation is significantly faster. It can decode 4,300 uptime cpu load event messages a second to C++ objects and encode 28,100 event messages a second from C++ objects. The Java implementation can decode 600 event messages a second and encode 21,900 event messages a second.

7. Conclusions and Future Work

This document describes an XML-based protocol for the transmission of performance events in a distributed environment. The protocol we describe is a proposed standard in the Performance Working Group of the Grid Forum. The purpose of this protocol is to address the problem of providing performance information in a standard way so that different tools can provide and use such information. We require such interoperability in a computational grid when we wish to analyze the performance of an application that uses several different resources.

We constructed two independent implementations of this protocol that interoperate. One implementation is written using Java, and the other using C++. We found that the C++ implementation can decode messages significantly faster than the Java implementation but the encoding time is similar.

References

[1] "Condor High Throughput Computing," http://www.cs.wisc.edu/condor/.
[2] "Data elements and interchange formats - Information interchange - Representation of dates and times," International Organization for Standardization ISO 8601, 1998.
[3] "The DOE Science Grid," http://www-itg.lbl.gov/Grid.
[4] "The Globus Project," http://www.globus.org.
[5] "Grid Forum," http://www.gridforum.org.
[6] "Grid Forum Performance Working Group," http://www-didc.lbl.gov/GridPerf/.
[7] "The Legion Project," http://www.cs.virginia.edu/~legion/.
[8] "The NASA Information Power Grid," http://www.ipg.nasa.gov.

[9] "The National Computational Science Alliance," http://www.ncsa.uiuc.edu/access/index.alliance.html.
[10] "The National Partnership for Advanced Computing Infrastructure," http://www.npaci.edu/.
[11] "XML-RPC Home Page," http://www.xmlrpc.com.
[12] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1," The World Wide Web Consortium 2000.
[13] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, vol. 11, pp. 115-128, 1997.
[14] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure,".: Morgan Kauffmann, 1999.
[15] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. R. Jr., "Legion: The Next Logical Step Toward A Nationwide Virtual Computer," Department of Computer Science, University of Virginia CS-94-21, June, 1994 1994.
[16] D. Gunter and B. Tierney, "A Standard Timestamp for Grid Computing." In Proceedings of the Global Grid Forum 1, 2001.
[17] T. Howes, M. Smith, and G. Good, *Understanding and Deploying LDAP Directory Services*: MacMillan Technical Publishing, 1999.
[18] W. Johnson, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid." In Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, 1999.
[19] M. Litzkow and M. Livny, "Experience with the Condor Distributed Batch System." In Proceedings of the IEEE Workshop on Experimental Distributed Systems, 1990.
[20] A. Pope, *The CORBA Reference Guide*. Reading, MA: Addison-Wesley, 1998.
[21] W. Stallings, *SNMP, SNMPv2, and CMIP: The Practical Guide to Network-Management Standards*. Reading, Massachusetts: Addison-Wesley, 1993.

A. Appendix

This appendix provides the XML Schema representation of the information we have described in this paper. First, we provide the schema for events, including two example events. Second, we provide the schema for event parameters, including two example event parameter sets. Third, we provide the schema for our protocol.

A.1. Event Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Jerry C Yan (NASA Ames Research Center) -->
<xsd:schema targetNamespace="http://www.gridforum.org/Performance/events"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" xmlns:gfperf-
event="http://www.gridforum.org/Performance/events" xmlns:gfperf="http://www.gridforum.org/Performance"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:import namespace="http://www.gridforum.org/Performance"
schemaLocation="D:\user\wwsmith\GridForum\XMLSchema\base-ns.xsd"/>
    <xsd:complexType name="EventType" abstract="true" final="#all">
        <xsd:annotation>
            <xsd:documentation>The basic event schema.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="TimeStamp" type="gfperf:string"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
    <xsd:element name="Event" type="gfperf-event:EventType" abstract="true">
        <xsd:annotation>
            <xsd:documentation>The basic event element that should be extended.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="UptimeCPULoad" substitutionGroup="gfperf-event:Event">
        <xsd:annotation>
            <xsd:documentation>An event containing CPU load information obtained from the Unix uptime
command.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-event:EventType">
                    <xsd:sequence>
                        <xsd:element name="Load1" type="gfperf:float"/>
                        <xsd:element name="Load5" type="gfperf:float"/>
                        <xsd:element name="Load15" type="gfperf:float"/>
                        <xsd:element name="HostName" type="gfperf:string"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Ping" substitutionGroup="gfperf-event:Event">
        <xsd:annotation>
            <xsd:documentation>An event containing the round trip time between hosts measured by the Unix
ping.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-event:EventType">
                    <xsd:sequence>
                        <xsd:element name="SourceHostName" type="xsd:string"/>
                        <xsd:element name="TargetHostName" type="xsd:string"/>
                        <xsd:element name="RoundTripTime" type="gfperf:float"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

A.2. Event Parameters Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Warren Smith (NASA Ames Research Center) -->
<xsd:schema targetNamespace="http://www.gridforum.org/Performance/parameters"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" xmlns:gfperf-
param="http://www.gridforum.org/Performance/parameters" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xsd:complexType name="EventParametersType">
        <xsd:annotation>
            <xsd:documentation>The basic input parameters to a producer when requesting
event(s).</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Period" minOccurs="0">
                <xsd:complexType>
                    <xsd:simpleContent>
                        <xsd:extension base="xsd:integer">
                            <xsd:attribute name="units" type="xsd:string"/>
                        </xsd:extension>
                    </xsd:simpleContent>
                </xsd:complexType>
```

```
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="EventParameters" type="gfperf-param:EventParametersType" abstract="true">
        <xsd:annotation>
            <xsd:documentation>The basic input parameters element that should be
extended.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="UptimeCPULoad" type="gfperf-param:EventParametersType" substitutionGroup="gfperf-
param:EventParameters">
        <xsd:annotation>
            <xsd:documentation>The parameters that can be passed to a producer when requestiong an
UptimeCPULoad event.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="Ping" substitutionGroup="gfperf-param:EventParameters">
        <xsd:annotation>
            <xsd:documentation>The parameters that can be passed to a producer of Ping
events.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-param:EventParametersType">
                    <xsd:sequence>
                        <xsd:element name="TargetHostName" type="xsd:string"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

A.3. Protocol Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Jerry C Yan (NASA Ames Research Center) -->
<xsd:schema targetNamespace="http://www.gridforum.org/Performance/protocol" xmlns:gfperf-
protocol="http://www.gridforum.org/Performance/protocol" xmlns:gfperf-
event="http://www.gridforum.org/Performance/events" xmlns:gfperf-
param="http://www.gridforum.org/Performance/parameters" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:import namespace="http://www.gridforum.org/Performance/events"
schemaLocation="D:\user\wwsmith\GridForum\XMLSchema\events-ns.xsd"/>
    <xsd:import namespace="http://www.gridforum.org/Performance/parameters"
schemaLocation="D:\user\wwsmith\GridForum\XMLSchema\params-ns.xsd"/>
    <xsd:complexType name="Request">
        <xsd:annotation>
            <xsd:documentation>The basic template for a request message.</xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="requestID" type="xsd:integer" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="Reply">
        <xsd:annotation>
            <xsd:documentation>The basic template for a reply message.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Return" type="xsd:string"/>
            <xsd:element name="ReturnDetail" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="requestID" type="xsd:integer" use="required"/>
    </xsd:complexType>
    <xsd:element name="SubscribeRequest">
        <xsd:annotation>
```

```
            <xsd:documentation>A message to request a subscription.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-protocol:Request">
                    <xsd:sequence>
                        <xsd:element name="SubscriptionID" type="xsd:integer"/>
                        <xsd:element ref="gfperf-param:EventParameters"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="SubscribeReply">
        <xsd:annotation>
            <xsd:documentation>A reply message to a subscription request.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-protocol:Reply">
                    <xsd:sequence>
                        <xsd:element name="SubscriptionID" type="xsd:string" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UnsubscribeRequest">
        <xsd:annotation>
            <xsd:documentation>A message to request that a subscription be cancelled.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-protocol:Request">
                    <xsd:sequence>
                        <xsd:element name="SubscriptionID" type="xsd:integer"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="UnsubscribeReply" type="gfperf-protocol:Reply">
        <xsd:annotation>
            <xsd:documentation>A reply message to an unsubscribe message.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="Event">
        <xsd:annotation>
            <xsd:documentation>A message containing an event that a producer sends to a
consumer.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Error" type="xsd:string" minOccurs="0"/>
                <xsd:element name="ErrorDetail" type="xsd:string" minOccurs="0"/>
                <xsd:element ref="gfperf-event:Event"/>
            </xsd:sequence>
            <xsd:attribute name="subscriptionID" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="QueryRequest">
        <xsd:annotation>
            <xsd:documentation>A message querying for an event.</xsd:documentation>
        </xsd:annotation>
```

```
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-protocol:Request">
                    <xsd:sequence>
                        <xsd:element ref="gfperf-param:EventParameters"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="QueryReply">
        <xsd:annotation>
            <xsd:documentation>A query reply message that contains an event.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-protocol:Reply">
                    <xsd:sequence>
                        <xsd:element ref="gfperf-event:Event"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="EventNamesRequest" type="gfperf-protocol:Request">
        <xsd:annotation>
            <xsd:documentation>A message sent from consumer to producer that requests information about
provided message types.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="EventNamesReply">
        <xsd:annotation>
            <xsd:documentation>A reply to an available events message that contains information about events
provided by the producer.</xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="gfperf-protocol:Reply">
                    <xsd:sequence>
                        <xsd:element name="Event" minOccurs="0" maxOccurs="unbounded">
                            <xsd:complexType>
                                <xsd:simpleContent>
                                    <xsd:extension base="xsd:string">
                                        <xsd:attribute name="eventns" type="xsd:string" use="required"/>
                                        <xsd:attribute name="paramns" type="xsd:string" use="required"/>
                                    </xsd:extension>
                                </xsd:simpleContent>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```